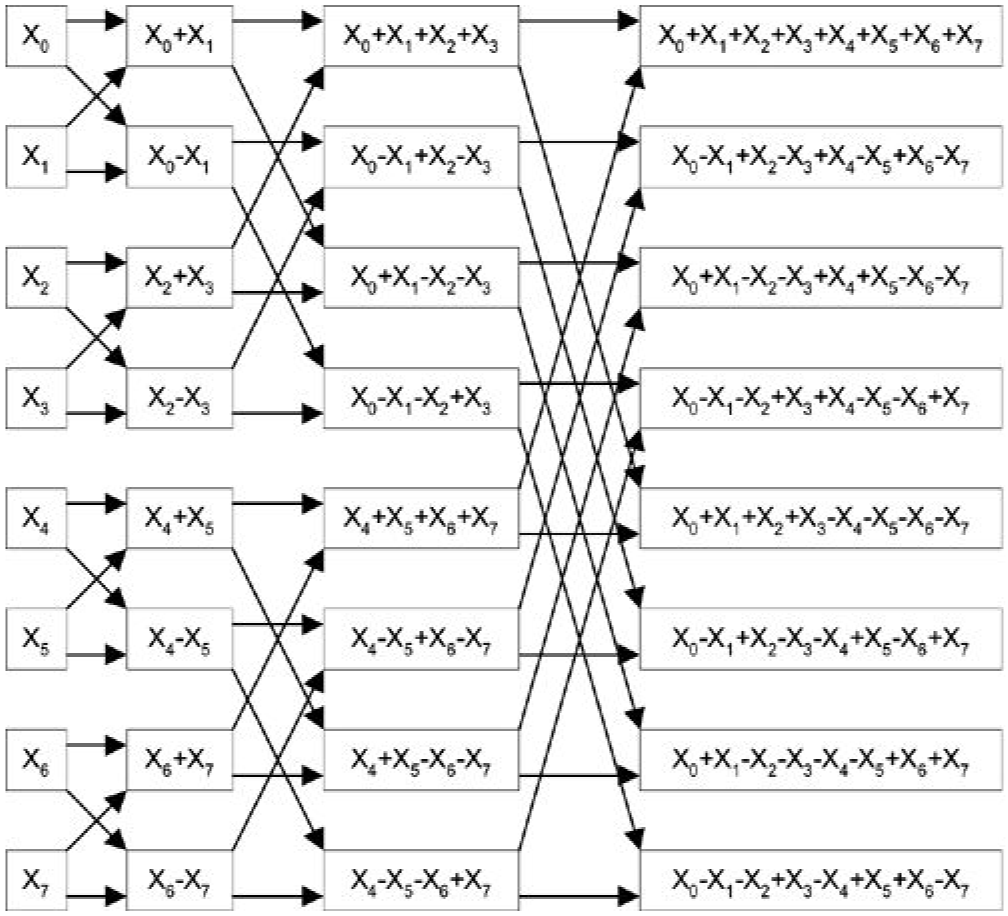
Efficient Neural Networks

A frozen neural network



X_0
X_1
X_2
X_3
X_4
X_5
X_6
X_7
X_0+X_1
X_0-X_1
X_2+X_3
X_2-X_3
X_4+X_5
X_4-X_5
X_6+X_7
X_6-X_7
$X_0+X_1+X_2+X_3$
$X_0-X_1+X_2-X_3$
$X_0+X_1-X_2-X_3$
$X_0-X_1-X_2+X_3$
$X_4+X_5+X_6+X_7$
$X_4-X_5+X_6-X_7$
$X_4+X_5-X_6-X_7$
$X_4-X_5-X_6+X_7$
$X_0+X_1+X_2+X_3+X_4+X_5+X_6+X_7$
$X_0-X_1+X_2-X_3+X_4-X_5+X_6-X_7$
$X_0+X_1-X_2-X_3+X_4+X_5-X_6-X_7$
$X_0-X_1-X_2+X_3+X_4-X_5-X_6+X_7$
$X_0+X_1+X_2+X_3-X_4-X_5-X_6-X_7$
$X_0-X_1+X_2-X_3-X_4+X_5-X_6+X_7$
$X_0+X_1-X_2-X_3-X_4-X_5+X_6+X_7$
$X_0-X_1-X_2+X_3-X_4+X_5+X_6-X_7$

Figure 1. A set of weighted sums with weights +1,-1.

The weighted sums in figure 1, acted on by an activation function form a neural network layer.
A frozen layer though. There is nothing to adjust.
However it is much faster than a conventional neural network layer.
First, no multiplies are needed for the weighted sums, just additions and subtractions.
Second, the overall number of operations needed for the weighted sums is cut from the n^2 required in a conventional layer to n*log2(n). In the example from 8*8=64 to 8*3=24.

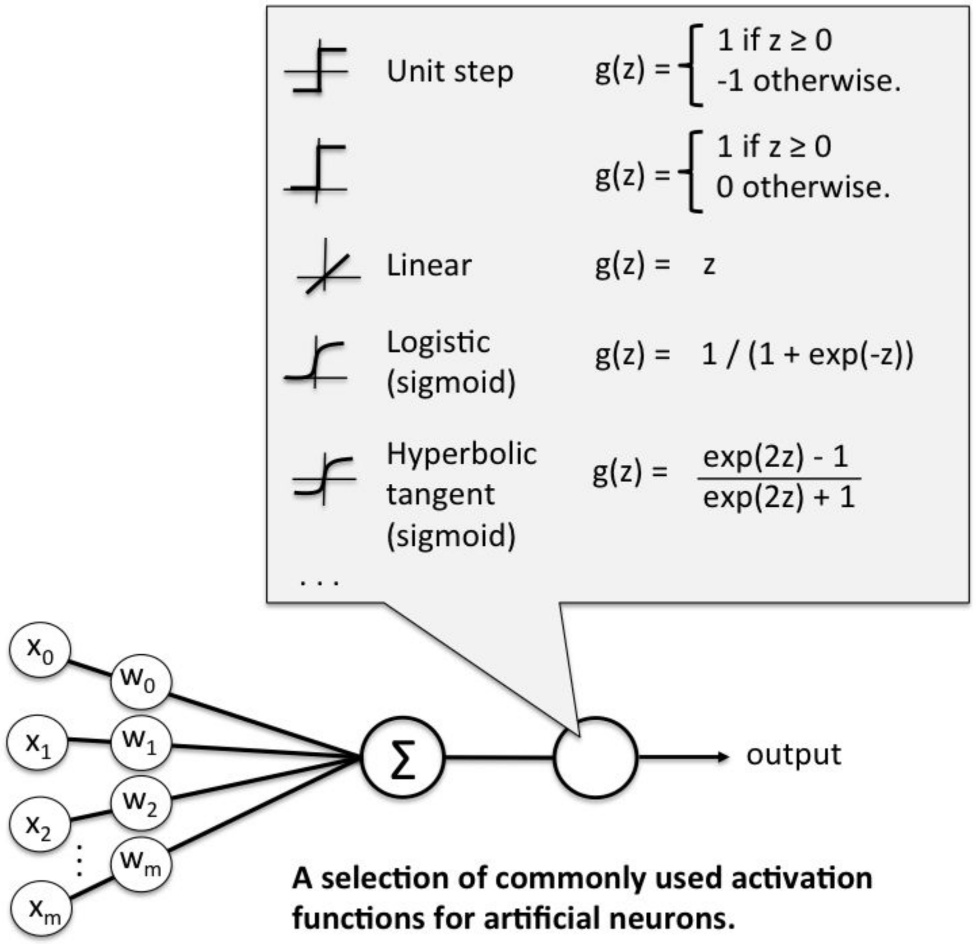

Unit step
$g(z) = \begin{cases} 1 \text{ if } z \geq 0 \\ -1 \text{ otherwise.} \end{cases}$
$g(z) = \begin{cases} 1 \text{ if } z \geq 0 \\ 0 \text{ otherwise.} \end{cases}$
Linear
$g(z) = z$
Logistic (sigmoid)
$g(z) = 1 / (1 + \exp(-z))$
Hyperbolic tangent (sigmoid)
$g(z) = \frac{\exp(2z) - 1}{\exp(2z) + 1}$
. . .
x_0
x_1
x_2
x_m
w_0
w_1
w_2
w_m
Σ
output
A selection of commonly used activation functions for artificial neurons.

Figure 2. A weighted sum acted on by an activation function.

Unfreezing the network

Static activation functions are used in conventional neural networks. There is nothing to adjust. It is possible to have parameterized activation functions that you can individually adjust. For example $g_i(z)=a_i*z$ $z>=0$, $g_i(z)=b_i*z$ $z<0$. When z is greater than zero it is multiplied by one parameter, when less than zero by a different one.
Using such parameterized activation functions with the transform in figure 1 allows you to unfreeze the network.

The math of the dot product

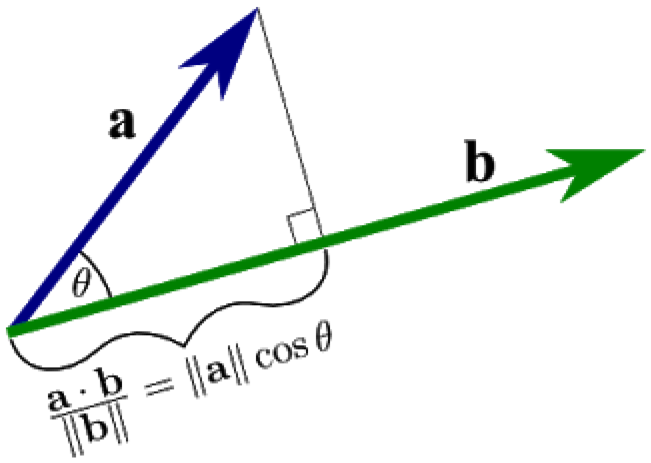
The weighted sum, also called the dot product has the following basic mathematical properties:

Algebraic

$$\mathbf{a}\cdot\mathbf{b}=\sum_{i=1}^{n}a_ib_i=a_1b_1+a_2b_2+\cdots+a_nb_n$$

Geometric

$$A \cdot B = |A| \cdot |B| \cdot \cos Ø$$


a
b
θ
$\frac{\mathbf{a}\cdot\mathbf{b}}{\|\mathbf{b}\|}=\|\mathbf{a}\|\cos\theta$

$$Ø = \arccos\left(\frac{A \cdot B}{|A| \cdot |B|}\right)$$

Statistical

Variances of Linear Combinations

If X, Y, . . . are ***independent*** random variables and

$$L = aX + bY + \ldots + c \text{ then}$$

Variance(L) = a^2 Variance(X) + b^2 Variance(Y) + ...

$$\sigma_L^2 = a^2\sigma_X^2 + b^2\sigma_Y^2 + \cdots$$

Most common:

Variance($X + Y$) = Variance(X) + Variance(Y)

Variance($X - Y$) = Variance(X) + Variance(Y)

The constant c has no effect on the variance

Central Limit Theorem for Sums

- If the collection of all random samples of size n are taken from a population having mean μ and standard deviation σ, then the sampling distribution of the sums will have mean $n\mu$ and standard deviation $\sqrt{n}\sigma$.
- If the population is normally distributed or if the sample size is large, then the distribution of the sums is also approximately normal.

Simultaneous Equations

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m , \end{aligned}$$

$$\underbrace{\begin{pmatrix} 2 & 3 & -4 & 1 \\ 1 & -3 & 1 & -1 \\ -2 & -1 & -3 & 1 \\ 1 & 1 & 1 & -3 \end{pmatrix}}_{\text{coefficient matrix}} \overbrace{\begin{pmatrix} w \\ x \\ y \\ z \end{pmatrix}}^{\text{unknown vector}} = \underbrace{\begin{pmatrix} 9 \\ 5 \\ -11 \\ -4 \end{pmatrix}}_{\text{result vector}}$$

The Walsh Hadamard transform

The transform in figure 1 is the fast in-place Walsh Hadamard transform (WHT) which has an equivalent matrix form.

$$H_2 = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}$$

$$H_3 = \frac{1}{2^{\frac{3}{2}}}\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{pmatrix}$$

$$(H_n)_{i,j} = \frac{1}{2^{\frac{n}{2}}}(-1)^{i\cdot j}$$

A scaling factor is all that is needed to keep vector length unchanged and then it is self-inverse. Obviously the rules of linear combinations of random variables and the central limit theorem will apply cleanly when it is used as a system of dot products.

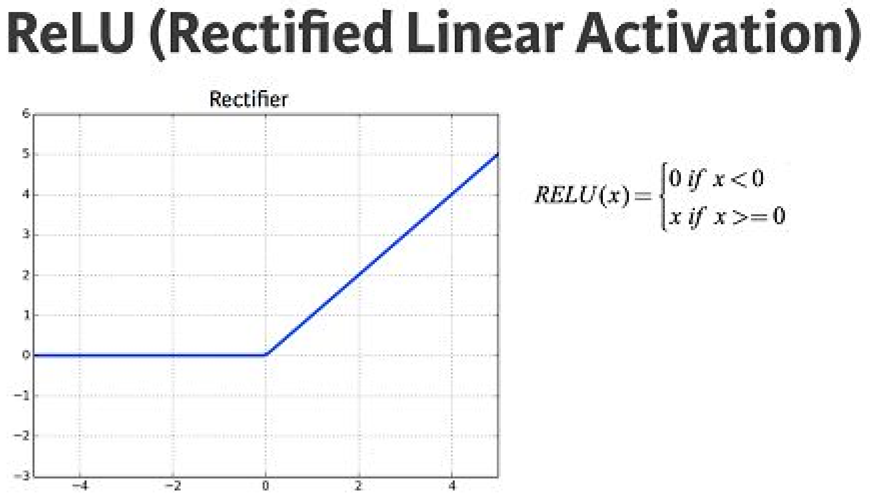
You can convert a column vector x of non-random data into random one by randomly flipping the signs of the data (perhaps in a predetermined pattern.)
Which is equivalent to multiplying by a diagonal matrix D with random +1,-1 entries (Dx). Then HDx is a random projection where you would expect the output elements to have 0 mean and a Gaussian distribution. The quality of the random projection can be improved if necessary eg. HD_2HD_1x.

Associative memory

Applying HD a number of times with different D gives a collection of random projections. After applying a nonlinear function to all the elements of all the projections you can partition those up

into inputs for some normal weighted sums. The pre-treatment allows the normal weighted sum to act as an effective associative memory.
If you choose a good nonlinear function such a bipolar binarization (+1,-1) then the training algorithm can be as simple as adding or subtracting an equal fragment of the error to each weight such that the recall error becomes zero.
By repeatedly learning a number of <pattern,response> items you effectively solve the related system of simultaneous equations.
If you learn only one <pattern,response> item the pattern vector and the weight vector have the same direction. By the linear combination of random variables rule there will be strong error correction to noisy pattern vectors.
If you store 2 <pattern, response> items the angles between the pattern vectors and the weight vector are non-zero. The length of the weight vector must increase to still get the wanted responses. This amplifies noise in the pattern vectors and reduces error correction.
The utility of the error correction is reduced by crosstalk for input patterns that are not fully orthogonal, meaning you cannot keep reapplying it to an autoassociative memory and hope to drive error to zero. Though you may use random intermediate patterns to help solve that.
The capacity is determined by being able to solve the simultaneous equations and for smaller binarized systems by quantization effects.
Used over capacity recalled values will be contaminated by Gaussian noise.
Note that any input pattern that has a smaller angle to the weight vector than any training pattern is a supernormal stimulus that is likely to produce a supernormal response. And suggests in neural networks that dot product outputs should be clipped to the limits observed from the training data to help avoid adversarial attacks.

ReLU as a literal switch



The ReLU activation function often used in neural networks.
An electrical switch is n volts in, n volts out when on. Zero volts out when off.

The ReLU activation function is x in x out when on. Zero when off, making ReLU an actual switch that throws itself on or off depending on conditions.
The weighted sum (dot product) of a number of weighted sums is still a linear system.
For a particular input to a ReLU neural network all the switches are decidedly in either the on or off state. A particular linear projection is in effect between the input and output.
For a particular input and a particular output neuron there is a particular composition of weighted sums in effect that may be condensed down into a single equivalent weighted sum of the input.
For a particular input and a particular ReLU: during forward propagation it is fed a particular composition of weighted sums that it makes a decision on, that may be condensed down into a single equivalent weighted sum of the input.

You can look at that to see what the network is looking at in the input or calculate some metrics like the angle between the input and the weight vector of the equivalent weighted sum.
Since ReLU switches at zero there are no sudden discontinuities in the output of an ReLU neural network for gradual change in the input. It is a seamless system of switched (commuted) linear projections.
The parameterized activation function used to unfreeze the WHT based neural network also creates a very similar system of switched (commuted) dot products.
Making it not so dissimilar from conventional neural networks. Wide versions of the unfrozen network work well for recurrent neural networks.